RL-TR-96-181
Final Technical Report
November 1996

CORBUS ENHANCEMENTS

BBN Systems and Technologies

Lisa Boone, Natasha Cherniack, Kobita Rajan, Sue Sours, Victor Voydock, and Edward Walker

DTIC QUALITY INSPECTED 2

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

19970224 090

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-96-181 has been reviewed and is approved for publication.

APPROVED:

ANTHONY M. NEWTON
Project Engineer

FOR THE COMMANDER:

JOHN A GRANIERO
Chief Scientist
Command, Control, & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/C3AB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Records, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-1188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1996	Final Jun 95 – Apr 96

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
CORBUS ENHANCEMENTS	C: F30602-95-C-0201 PE: 63728F PR: 2530
6. AUTHOR(S) Lisa Boone, Natasha Cherniack, Kobita Rajan, Sue Sours, Victor Boydack, Edward Walker	TA: 01 WU: P4

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
BBN Systems and Technologies 10 Moulton Street Cambridge, MA 02138	N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Rome Laboratory/C3AB 525 Brooks Road Rome, NY 13441-4505	RL-TR-96-181

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Anthony M. Newton/C3AB/(315)330-3097

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release: Distribution Unlimited	

13. ABSTRACT (Maximum 200 words)

The key objective for this contract was to improve three areas of the Corbus system: installation, persistent data storage, and the graphical tools which view and modify Corbus configuration information. Improving the installation process was accomplished at two levels. First, the Corbus hierarchy was reorganized in a manner which made it more difficult to install Corbus incorrectly. Second, the steps necessary to install Corbus have been reduced to a simple one step process. The persistent data storage improvements made it possible for the data managed by the Corbus servers to be moved from one machine to another, even if those two machines were running different operating systems. Finally, we improved the Corbus graphical tools so that they now use a toolkit which is supported on all the platforms that support Corbus.

14. SUBJECT TERMS		15. NUMBER OF PAGES
Distributed systems, CORBA, object-oriented, distributed computing, DCE, Cronus.		32
		16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	SAR	SAR

Table of Contents

List of Figures and Tables

1. Introduction

The key objective for this contract was to improve three areas of the Corbus system. These areas were installation, persistent data storage, and the graphical tools which view and modify Corbus configuration information.

Improving the installation process was accomplished at two levels. First, the Corbus hierarchy was reorganized in a manner which made it more difficult to install Corbus incorrectly. Second, the steps necessary to install Corbus have been reduced to a simple one step process.

The persistent data storage improvements made it possible for the data managed by the Corbus servers to be moved from one machine to another, even if those two machines were running different operating systems.

Finally, we improved the Corbus graphical tools so that they now use a toolkit which is supported on all the platforms that Corbus is supported on. Previously, the toolkit which we used to build the graphical tools would only run on SunOS 4.1.3; all other Corbus supported platforms would have to run a forms based version of the tool. While the forms based version was operational and supported all of the functions that the graphical tools did, it was a nuisance to have to learn two different programs in order to accomplish the same Corbus task.

2. Installation Improvements

We made a number of improvements to the Corbus installation process. We changed environment variables to accommodate the system name change (Cronus->Corbus), we modified the Corbus hierarchy to follow a more standard naming convention, we changed installation terms which had proved confusing to our users in the past, and we simplified the overall installation process. Each one of these improvements are discussed in detail in the following sections.

2.1 Environment variables

The first change we made was to accommodate the change of names in the Corbus system (from Cronus to Corbus). The environment variables CRONUS_ROOT and CRONUS_CONFIG were changed to CORBUS_ROOT and CORBUS_CONFIG respectively. CORBUS_ROOT represents where Corbus is installed (default */usr/corbus*) and CORBUS_CONFIG represents which configuration Corbus is installed in (default 0). For backwards compatability, we still provide support for the CRONUS_ROOT and CRONUS_CONFIG environment variables but we encourage all our users to change over to the new naming convention as soon as possible. We plan to remove the support for all Cronus environment variables in future releases.

If the environment variable CORBUS_ROOT is not set, Corbus will see if the environment variable CRONUS_ROOT is set and assume that value if it is. If neither CORBUS_ROOT or CRONUS_ROOT is set, Corbus will see if the directory */usr/cronus* exists and if it does, assumes that the Corbus system is installed in */usr/cronus*. If /usr/cronus does not exist, Corbus will assume that the system is installed in */usr/corbus*. Similarly for CORBUS_CONFIG, if this variable is not set, Corbus will take the value of CRONUS_CONFIG. If CRONUS_CONFIG is not set, Corbus will assume the default value of 0.

2.2 Corbus Hierarchy

The Corbus hierarchy initially consisted of the following directories: cetc, cbin, clib, mgrbin, mgrs, install, include, and public (see Figure 1). The first thing that we did was to change the names of some of the directories so that they would follow the same sort of naming convention that SunOS, Solaris and (commercial) off-the-shelf products use. Specifically, we changed cetc to etc, clib to lib, and cbin to bin. (For the remainder of this technical report, the new naming convention will be used.)

In addition, we wanted to address the issue of users who have multiple installations of Corbus on one machine. Previously, to install Corbus multiple times on one machine, users would have to set up private etc (cetc) and mgrs directories for the installation. These two directories contain log files, configuration information, and object databases which cannot be shared between

installations. Setting up the private directories was accomplished using the **extraconfignum** installation process. The **extraconfignum** installation did not install Corbus, rather it merely set up these private directories and named them *etc$CORBUS_CONFIG* and *mgrs$CORBUS_CONFIG*. The *etc$CORBUS_CONFIG* and *mgrs$CORBUS_CONFIG* directories were created by copying the etc and mgrs directories and editing some of the system start up files accordingly. When the **extraconfignum** installation was performed properly, it was very easy to take a look at the $CORBUS_ROOT directory and know instantly in what configurations Corbus was installed on that particular machine.
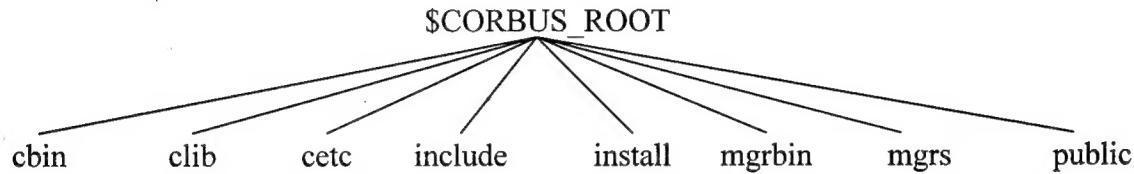

Figure 1: Original Corbus Hierarchy

The disadvantage to this scheme was that the initial cetc and mgrs directories could still be used for a Corbus installation. If either the cetc or mgrs directory were corrupted in any way, any further attempts to install Corbus in multiple configurations using the **extraconfignum** process would fail. Furthermore, a one-time glance at the $CRONUS_ROOT directory would no longer tell users what configuration(s) Corbus was installed in.

To remove this confusion, we decided to not allow Corbus to be installed in the etc and mgrs directory at all. Instead, we decided to use those directories as a shared repository of Corbus files for the extraconfignum installation and force all installations to perform the **extraconfignum** installation. The Corbus directory hierarchy was modified to make this change more intuitive. The new hierarchy removed the etc and mgrs directory and put them into a new directory called *shared*. (See Figure 2). The **extraconfignum** installation now works by copying the *shared/etc* and *shared/servers* directory, making the necessary modifications to the files, and names them *etc$CORBUS_CONFIG* and *servers$CORBUS_CONFIG*.
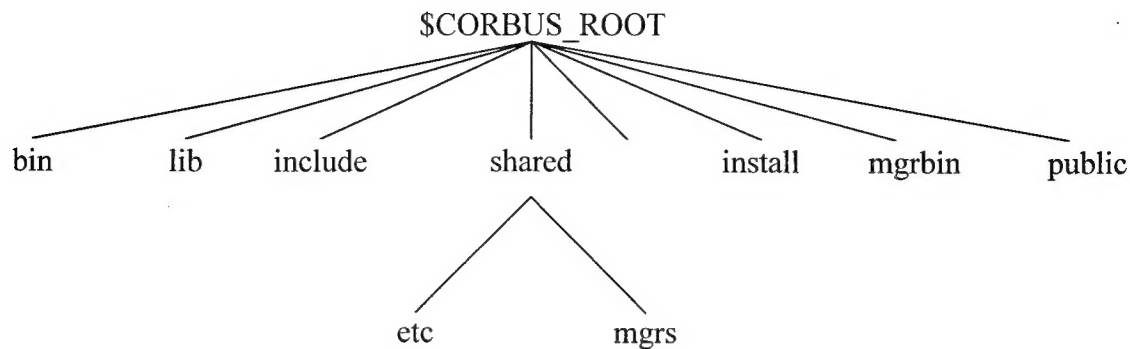
$CORBUS_ROOT

bin   lib   include   shared   install   mgrbin   public

etc   mgrs

Figure 2: New Corbus Hierarchy

2.2.1 Setting up the Corbus hierarchy

The first step in installing Corbus, is to set up a version of the Corbus hierarchy where you want Corbus installed. This could be accomplished three different ways: extracting Corbus from a tape/tar file, running the command *"cr install satellite"* and selecting the **extraconfignum** option, or by running the command *"cr install satellite"* and selecting the **satellite** option.

Extracting Corbus from a tape or tar file is straight forward, once the files have been put onto disk, the user proceeds with the installation instructions. Selecting the **extraconfignum** option when running the command *"cr install satellite"* was used when users want to install additional configurations of Corbus on a machine which already had Corbus installed on it (as discussed in section 2.2). Selecting the **satellite** option when running the command *"cr install satellite"* was generally used when trying to install Corbus on a diskless machine or when trying to install Corbus on a machine(s) which had access (via NFS) to a Corbus installation running the same operating system.

The term *"cr install satellite"* was very misleading to users. Corbus was never installed during the **satellite** or **extraconfignum** installation and the word satellite was non-intuitive. We have changed *"cr install satellite"* to *"cr install hierarchy"* to alleviate this confusion. The *"cr install hierarchy"* command no longer prompts you for what type of setup you want to do (satellite vs. extraconfignum), it assumes a satellite installation. The reason for removing the extraconfignum had to do with the steps we took in simplifying the installation process. This is explained fully in section 2.3.1.

4

2.3 Installation process

Previously to install Corbus, users had to follow a number of steps. They had to get a copy of Corbus onto disk and set up private etc and mgrs directories if they wanted to install the system multiple times. After setting up the Corbus hierarchy, users had to initialize Corbus system files by running a provided tool (*inituno*) and by editing another system file (hostfile) to inform Corbus of the new host (see Figure 3). After these steps had been performed, the Corbus kernel would be booted and the actual installation process could begin.
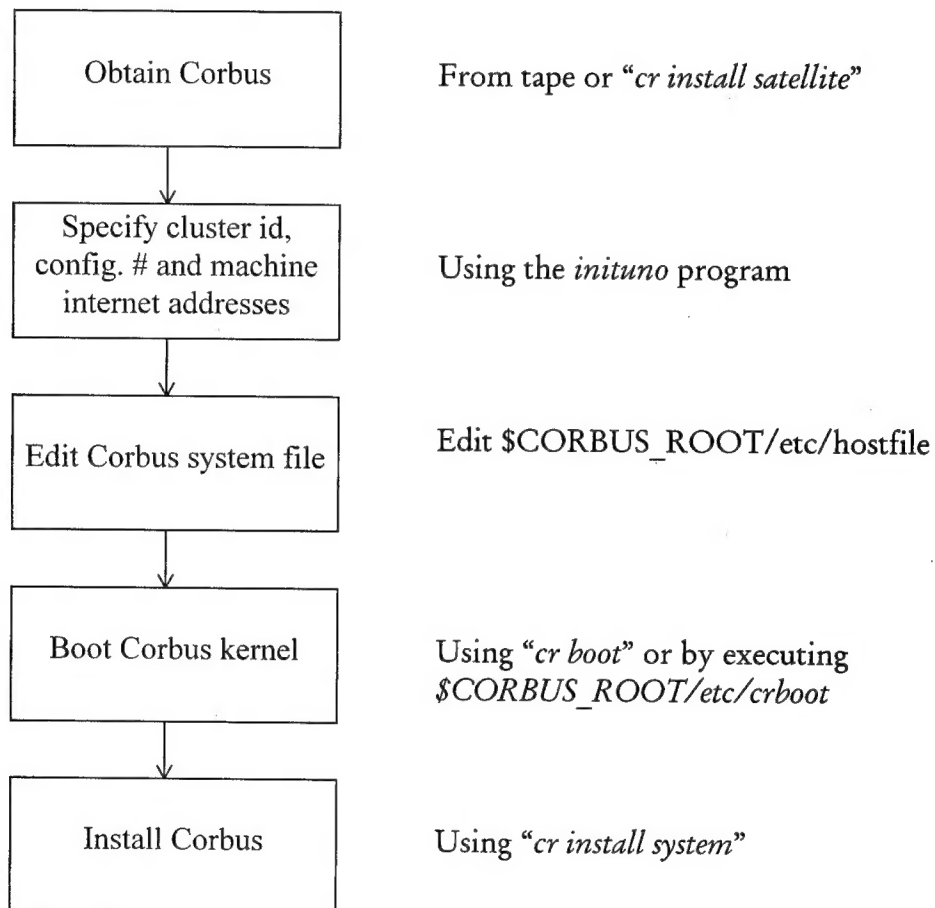
Obtain Corbus	From tape or "*cr install satellite*"
Specify cluster id, config. # and machine internet addresses	Using the *inituno* program
Edit Corbus system file	Edit $CORBUS_ROOT/etc/hostfile
Boot Corbus kernel	Using "*cr boot*" or by executing *$CORBUS_ROOT/etc/crboot*
Install Corbus	Using "*cr install system*"

Figure 3: Original Corbus Installation Process

There were a couple of problems to this approach. First, if any of the above steps were not performed, or performed incorrectly, then the installation process would fail with a single error

message. More often than not, users would get confused at the sight of the error message and immediately call the hotline or send an email message asking for assistance. Messages such as "LOCAL_IPC_FAILURE, Corbus kernel not running or in wrong configuration" albeit accurate, didn't help much if the person was unfamiliar with the Corbus system. Sure the message says Corbus kernel not running, but why? Seasoned Corbus users would know to look in the kernel's log file (k.log) for more information but novice users would have no idea where to look. One of the primary goals in improving the installation process was not only to simplify the steps needed to perform the installation, but also to provide informative error output if the installation failed.

Another problem with the installation process was encountered if the installation failed in any way, log files and databases were left behind and not cleaned up. If the user attempted to install the system again, unexpected questions from the installer such as "I did not expect to find authen.odb here. Rename to x_authen.odb?" confused the users. Ideally, if the installation fails, we want the user to try the installation again without having to clean up any files left behind from the failed attempt.

Finally, since so many of our Corbus users embedded Corbus as part of their own applications, they had to develop advanced installation instructions for all of their customers so that they could install not only their application, but Corbus as well. As part of this contract, we tried to make automating the installation of Corbus as simple as possible to ease the installation of large complicated applications which have Corbus embedded in them.

2.3.1 One-step installation

Our new installation process addresses the above problems. Now, instead of having to run multiple steps to install Corbus, users need only perform two actions to install: set up the Corbus hierarchy and install Corbus. The new installation process combines all previously required steps and performs them seamlessly using the command *"cr install system"* (see Figure 4).
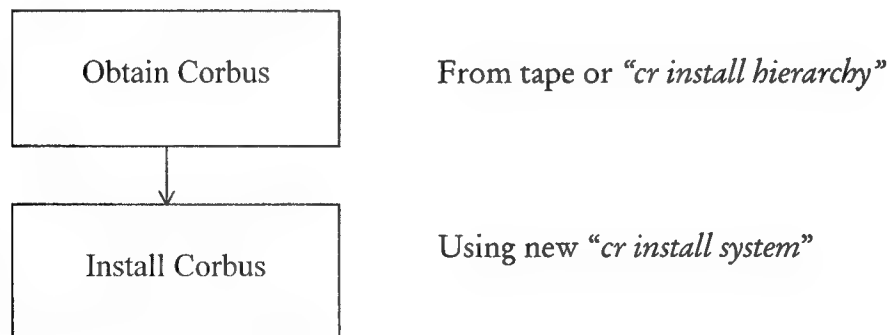

| Obtain Corbus | From tape or *"cr install hierarchy"* |

| Install Corbus | Using new *"cr install system"* |

Figure 4: New Corbus Installation Process

When the command *"cr install system"* is run, a number of things happen. First, an extraconfignum setup is performed to set up the private etc and servers directories. The new installer now forces all users to install Corbus in the etc$CORBUS_CONFIG and servers$CORBUS_CONFIG directories to avoid confusion. It is now impossible to install Corbus in etc and servers, not only because the installer sets up the private etc and servers directory automatically, but because the etc and servers directories have been moved to $CORBUS_ROOT/shared/etc and $CORBUS_ROOT/shared/servers (as discussed in section 2.2). We believe that this arrangement will reduce confusion for users wishing to install Corbus in multiple configurations. Once the private etc and servers directories have been set up, the installation continues by creating and editing the necessary system files *($CORBUS_ROOT/etc$CORBUS_CONFIG/UNO$CORBUS_CONFIG.DAT,* *$CORBUS_ROOT/etc$CORBUS_CONFIG/hostfile).* Previously, users would do this manually by using the inituno program and their local editor. The new installer has added the inituno functionality and will make the necessary editing changes to the hostfile. At this time, the installer will boot the kernel and begin the installation process.

At any point, if an error is encountered, the installer will abort the installation and an informative error message will be written to *stdout* and to the file /tmp/corbus.errors indicating what step failed. If a novice Corbus user encounters an error during installation, they can now take the generated file and send it to a more experienced user to diagnose. The file */tmp/corbus.errors* will not only state where the installation failed, but it will also offer a collection of pointers where one would look to remedy the problem.

When an installation fails, all evidence of the attempted install will be removed from the machine so that the user can type *"cr install system"* and try to install Corbus again. We found this to be a necessary step since history demonstrated that if something fails, users will want to try it again by running the same procedure. Unless the problem which caused the first installation attempt to fail was fixed, the output from trying to install a second time should be identical to the previous attempt and will be due to the new installation procedure.

2.3.2 Informative output

However, sometimes it is necessary to leave information of the failed installation attempt around. For instance, the kernel's log file can contain some useful information as to why it was unable to boot properly, hence the failed installation. Cleaning up after a failed install will remove this log file and leave the user with no real useful information to diagnose. A new command line flag has been added to the installer which will cause any failed installation attempt to not cleanup. This way, the user can take a look at the files left behind, remedy the problem, and try to install the system again. To run the installer so that it doesn't clean up after a failed installation attempt, the user should type *"cr install /nocleanup system"*. Once the problem has been diagnosed and fixed, the files left behind from the failed installation will need to be removed. This is done by running the supplied script *scrub* which is located in *$CORBUS_ROOT/install/scrub*. Running this script will remove all evidence that Corbus was ever installed on a machine (whether or not the installation was successful).

2.3.3 Automating installation

Due to the new advanced features of the Corbus installer, there are more questions that the user must answer during the installation process (such as broadcast address, netmask address, cluster identifier). The questions are very straight forward but could be confusing to users who aren't expected to use Corbus directly. Take for example a complicated system which has Corbus embedded in it. To ship and install this system, users will be required to know not only about the application itself, but enough about Corbus so that they can install the complete system. To address this issue, we provided a way to automate the entire Corbus installation based on a Corbus configuration file. This file is shipped with Corbus and is located in *$CORBUS_ROOT/install/corbus-config-template*. Users wishing to automate the installation of Corbus simply copy the file corbus-config-template to another file, edit the appropriate fields, and run the command *"cr install /input=corbus-config-filename system"*. This way, any application which ships Corbus as part of their system, can fill out this configuration file properly and ship it along with the application. User's who install the application need know nothing about Corbus or how to install it and can type one line to complete the Corbus installation process.

We have also provided a tool, *autofill*, which will fill out as many of the fields possible in the configuration file. Information such as the name of the machine, internet, network and broadcast address of the machine can be determined via a few simple system calls. Information which can be guessed, such as where to install Corbus or in what configuration and cluster to install Corbus in, will be set to the default values (*/usr/corbus*, 0, and 1 respectively) and can be changed manually by the user after *autofill* has completed but before *"cr install /input=corbus.config system"* is run. Information which can't be determined at all such as what type of installation to perform (additional or pristine) or a textual decription of the machine to install Corbus on, will be left the same as the *corbus-config-template* file and will require the user to edit the file after *autofill* is complete. Autofill, by default will write this configuration information to standard out. The file */output* will direct the output to a file instead (e.g. *autofill /output=my-corbus-config*)

2.4 Miscellaneous fixes

In addition to the above mentioned installation improvements, we also made some minor modifications to the installer which made the Corbus installation easier to use and understand. First, we updated the file template.ins which is used when new Corbus servers are developed and users want to add it to the recognized list of servers which can be installed using the *"cr install system"* command. The new version of template.ins has removed all Cronus-isms, recognizes the new Corbus hierarchy, and has better instructions for users.

Second, we fixed the *"cr install hierarchy"* (previously *"cr install satellite"*) command so that all the files are copied to the new hierarchy. Prior to this fix, only the directory names and a portion of the files were being copied and users had to manually copy the missing files.

Finally, we changed the installer so that you no longer have to be in the $CORBUS_ROOT/install directory to run it. This decision was made based on user's problems when the Corbus hierarchy was set up using the "cr install hierarchy" command. "cr install hierarchy" (as discussed in section 2.2.1) sets up the Corbus hierarchy by creating symbolic links to shared directories (public, install, lib, include, bin), and creating the private directories locally (etc, servers). Due to this setup, it's very easy to accidentally end up in the wrong Corbus hierarchy by typing "cd ../install" from the etc$CORBUS_CONFIG directory. Instead of ending up in the $CORBUS_CONFIG/install directory, due to the symbolic links, you will end up in the satellite server's Corbus hierarchy. Running the installer from the wrong install directory could lead to a lot of confusion when trying to debug a failed installation attempt. Additionally, we felt that it was unnecessary to require that the installation be run from the install directory. The new installer will read the installation scripts from $CORBUS_ROOT/install thereby allowing users to run the Corbus installation from any directory.

3. Persistent Data Storage

All Corbus servers, both those which are distributed with the system and those which are developed using the Corbus Server Development Toolkit, are responsible for managing a collection of objects. The capabilities (methods) of these objects are defined in the .idl file and the representation of the actual object is defined in the .server file.

A typical operation invoked on an object is implemented the following way by the server: the current state of the object which the operation was invoked on is retrieved, the object is manipulated as dictated by the operation, and the new version of the object is stored. All Corbus objects are stored in a database (which is currently implemented as a flat file) on disk which is accessible by its corresponding server only. There are two pieces of information associated with an object representation: the object data itself and the object descriptor. The object data itself is server dependent and is specified in the .server file. The object descriptor is server independent and stores information such as the date of creation/modification of the object and access control information. Different Corbus servers cannot share databases due to the fact that the different servers would expect the object data to be formatted in the same way. Multiple versions of the same Corbus server should be able to share databases regardless of what platforms the servers have been compiled for.

When the persistent data storage scheme was first developed, Corbus wrote the object data to disk in a canonical format. That is to say, the object data was written in such a way that any process could read it regardless of what operating system it was running. This canonical format is the same sort of mechanism used in message passing in Corbus; it provides a way for different operating systems the capabilities of deciphering the same data. Theoretically then, a Corbus server's object database could be moved from one machine to another and no additional work would need to be done. Unfortunately, the object's descriptor was not stored in canonical format so any attempts to move an object database to a different machine running a different O.S. would not work. Moving a database from one machine to another would be desirable if there were anticipated power outages for a specific machine and you wanted to have availability of a server which was running on that machine.

We changed how the object descriptors are stored so that now they follow the same format as the object data itself. All the contents of an object database is now stored in canonical format allowing the migration of a database from one machine to another. If a user wanted to move a (non-replicated) server to run on a different machine, they need only copy the object database to the new machine, compile the Corbus server for the new machine, and start up the server. It's important to note that replicated servers can never move their object databases from one machine to another. Replicated servers store specific host location information in the object descriptor which is used to update the object's replicas. Moving replicated databases from one machine to another will break the Corbus replication strategy and cause the replicated servers to not only fail to start, but possibly corrupt the object databases as well.

4. Graphical Tools

The Corbus graphical tools have been ported from the diamond toolkit, a BBN developed graphical toolkit, to the TK/TCL toolkit. We did this because the diamond toolkit was not supported on the Solaris or HP platform. This lack of support required the need for an additional(non-graphical) tool to supply the functionality of the **examine cluster tool**. The **examine configuration tool** was never supported for either Solaris or HP platform. The TK/TCL toolkit is supported on all of the current platforms that Corbus is supported on. We are now able to provide a common graphical user interface for the **examine cluster tool**, as well as the **examine configuration tool** on all platforms that Corbus is supported on.

The examine tools have the same basic layout. Each tool is contained in a dialog box with a top-level menu at the top of the dialog box. Each tool displays information about the current configuration via a grid. The **examine configuration tool** displays Host/Service information. The **examine cluster tool** displays Cluster/Service (exports, imports, domains) information.

The new examine tool works basically the same as the old tool. There are some subtle differences to make the tools a little more consistent with each other. A left mouse click over the top-level menu buttons will perform the expected action. A left mouse click over any of the labels on the grids will pop up an informational dialog box about the selected item. Holding down the right mouse button over any of the grid squares will pop up an action menu for that tool.

The examine tool is brought up in the same manner as the old tool. The operator must login and then type:

      cr examine *clust | config*

This will bring up the selected examine tool. The information displayed will reflect the current configuration.

There are detailed man pages in the Corbus Users Manual.

c

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

    a. Conducts vigorous research, development and test programs in all applicable technologies;

    b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

    c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

    d. Promotes transfer of technology to the private sector;

    e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.